Distributed Architecture for Programming Environments

Anne-Marie Dery, Laurence Rideau

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Distributed Architecture for Programming Environments

Anne-Marie Dery, Laurence Rideau

N° 2918

Juin 1996

THÈME 2

ISSN 0249-6399

Distributed Architecture for Programming Environments

Anne-Marie Dery, Laurence Rideau

Thème 2 — Génie logiciel
et calcul symbolique
Projet CROAP

Rapport de recherche n ° 2918 — Juin 1996 — 17 pages

Abstract: We present a technology used to transform a monolithic programming environment into a heterogeneous distributed system. We propose a communication protocol adapted to message passing between components in this new architecture. This protocol allows structured data transfer and remote operations. It has been implemented and is used in the CENTAUR programming environment generator.

Key-words: Distributed Applications, Structured Data Transfer, Message Protocol, Programming Environments

(Résumé : tsvp)

UNSA-CNRS, Email: pinna@essi.fr
Email: Laurence.Rideau@inria.fr

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Environnements de Programmation Distribués

Résumé : Nous présentons une méthode de transformation d'un environnement de programmation monolithique en un système hétérogène distribué. Nous proposons un protocole de communication adapté à la communication de messages entre les composants de cette nouvelle architecture. Ce protocole permet le transfert de données structurées et l'exécution d'opérations à distance. Il a été implémenté et est couramment utilisé dans le générateur d'environnements de programmation CENTAUR.

Mots-clé : Applications Distribuées, Transfert de Données Structurées, Protocole de Messages, Environnements de Programmation

1 Introduction

Distributed systems began to appear in the 1970s, following the availability of computer networks. Then, the key to computer technology has been information gathering and processing [26, 10]. The idea of distributed programs is relevant to many areas of computing, indeed all of them. We can find distributed database systems, distributed artificial intelligence systems, distributed software engineering tools, etc. Programming environments easily lend themselves to distributed computing. Initially, such systems were monolithic, and, therefore replacing components or integrating a tool developed elsewhere was difficult. However, such systems can easily be viewed as a collection of tools working on abstract syntax trees. They provide functionalities such as parsing, editing, type-checking, and compiling. Parsers create the data, editors visualize it, type-checkers type check it, etc. While a system's architecture can be done, so that it is modular, and the system is open, so that new tools may be added and components made reusable, this is not sufficient in the long run. With the whole system running as a single process, where all modules are linked together, the size of the process can become excessively large. Moreover, a user may only need part of the system, but the removing, or modifying, of a module then implies the re-building of a specific executable. Specifying modules and distributing them has the advantage of enlarging the degree of parallelism, tools can run on different processes on different workstations, and also allows one to implement each component or tool with the best adapted programming language. Distributed programming gives a light weight solution to the problem of interoperating several programming paradigms (functional programming, logic programming, imperative programming, etc.). Furthermore, a distributed architecture facilitates the use of a component (or a subset of components) independently of the others and the integration of new components. The distributed application is then open to the external world.

In this paper, we explain why we think it is important to distribute programming environments and why we think it is necessary to propose an adapted communication protocol between components. We give then an overview of this protocol named *message protocol*, we explain the advantages of such a high-level protocol, splitting large data to preserve atomicity of transferred elements and to allow interruptions during transfer, and finally we describe its integration in a programming environment. This application protocol gives some guidelines for choosing a well-adapted *communication platform* without being dependent on a particular one. In section 6, we discuss our choice of communication platform. The *message protocol* has been implemented and validated; it is used to distribute the programming environments generated by the CENTAUR[1] system [6, 8, 17]. The use of Centaur leads us to implement programming environments that are especially well-suited to a variety of programming languages. In particular, we study tools for the ML language (functional programming), the Eiffel language (object-oriented programming), the Sisal programming language (parallel data-flow programming), or C (imperative programming) [21, 1, 2]. Mo-

[1] Centaur was developed under the European Community's Esprit program in the projects: "Generation of Interactive Programming Environments", Gipe (1984–1989) and Gipe II (1989–1993).

reover the Centaur system is currently used to allow the communication between external components and Centaur programming environments (see section 7, for more details).

2 Motivation

In this section, we specify more precisely why modularity is not sufficient for programming environment implementation and why we are interested in building distributed programming environments.

Advantages of a distributed programming environment. A distributed architecture enforces autonomy, reuse of components, and maintenance capabilities:

- In a monolithic programming environment, specific components such as a parser, or a pretty printer, are not easily reusable outside of the system. In a distributed architecture, each process is able to function independently. In this context, each component has to be provided with an appropriate interface to ensure its *autonomy*; in return it may have to collaborate with other components to perform its task.
- In modular architectures, when modules are implemented with different programming languages, they must be adapted to interact with the other modules. This is an obstacle to the *reuse* of these modules. This difficulty is reduced in distributed architectures. The protocol definition clearly specifies how a component can be connected to another. All the entities that are faithful to the same communication protocol can interact independently of their own implementation.
- In a monolithic application, changing the version of one of the components (bug fixing or functional extensions) implies the generation of a new version for the complete application. However, the *maintenance* becomes easier when the application is distributed because only the concerned component is modified. Furthermore the application protocol ensures the global system consistency.

A distributed architecture allows one also to introduce parallelism in tasks and to facilitate the cooperation between several development teams.

A distributed architecture for a programming environment. Since the separation of processes (servers and clients) by a network introduces the possibility of performance degradation due to communication delays, a study must be done on the placement of the different modules. In the context of programming environments, Clement [8] proposes to distribute programming environments along functional lines: each service is managed by a separate component, namely, *parser*, *pretty-printers*, *evaluators*, *compilers*, and *type-checkers*. Some of these components are themselves modular. A *pretty-printer* has two internal modules: a tree compiler and a graphical machine. These two modules may be separated in two processes. Each component provides a specific service, and can make calls to different